R·I·T

Computer Science II
4003-232-08 (20082)

Week 1: Review and Inheritance

Richard Zanibbi
Rochester Institute of Technology

Review of CS-I

Syntax and Semantics of Formal (e.g. Programming) Languages

Syntax

The rules defining legal statements and their combination

e.g. Rules for valid Rochester area phone numbers

e.g. Rules for constructing legal Java programs

Semantics

The meaning of statements in the language (what statements *represent*). Often depends on *context* of use

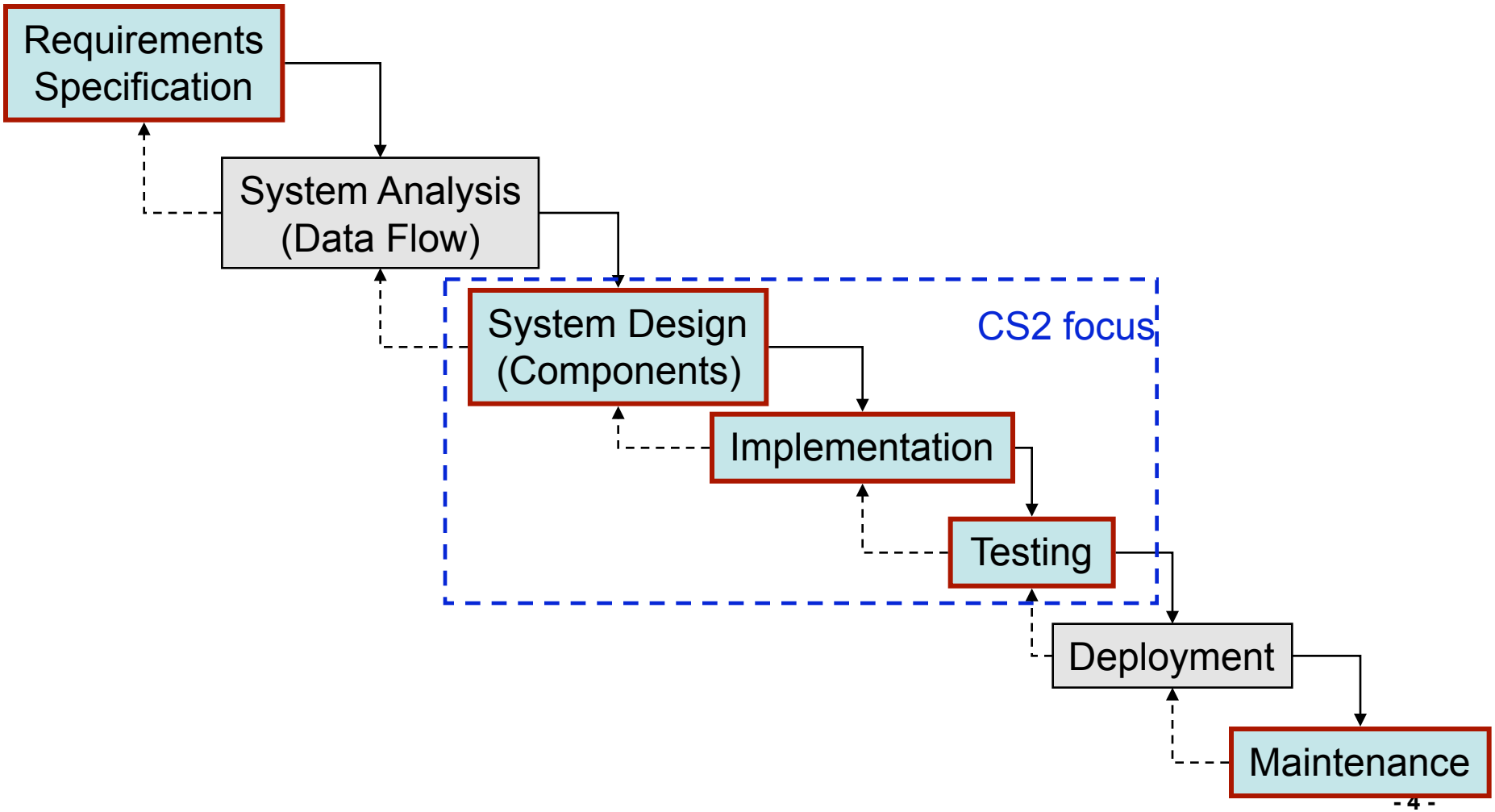
e.g. What does the following represent:

"349-5313"

e.g. What does the following Java statement represent:

```
System.out.println(x);
```

A “Waterfall” Model of Software Development



Miscellaneous CS1 Java Topics

Primitive Data and Operations

- Variable declaration, initialization, and assignment
  - Constants ('final')
  - Arguments (pass-by-value) and local variables
- Floating point vs. integer arithmetic
- Type conversions (widening and narrowing), and casting
- Operator precedence and associativity (Liang Appendix C)

Object-Oriented Features

- Class and method syntax: constructors, return types (e.g. void, int, double)
- The new operator (instantiates objects from classes)
- Visibility modifiers (public and private)

Variable Properties

1. **Location (in memory)**
2. **Identifier (name)**
   - A symbol representing the location
3. **Type (of *encoding* used for stored data)**
   - Primitive (e.g. int, boolean), or
   - Reference (address in memory of a class instance (object))
4. **Value**
   - The primitive value or reference (for objects) stored at the variable location

Memory Diagrams: Illustrating Variable Properties

Variable Storage (Memory Locations)

Represented using a box

Variable Identifiers and Types

Indicated using labels outside the box (e.g. x : int)

For static variables, indicate 'static' and class name

- e.g. x : int (static Widget)

Variable Values

– Primitive types: show value in the box (e.g. for integers, show decimal value)

– Reference variables: draw arrow from box to object data

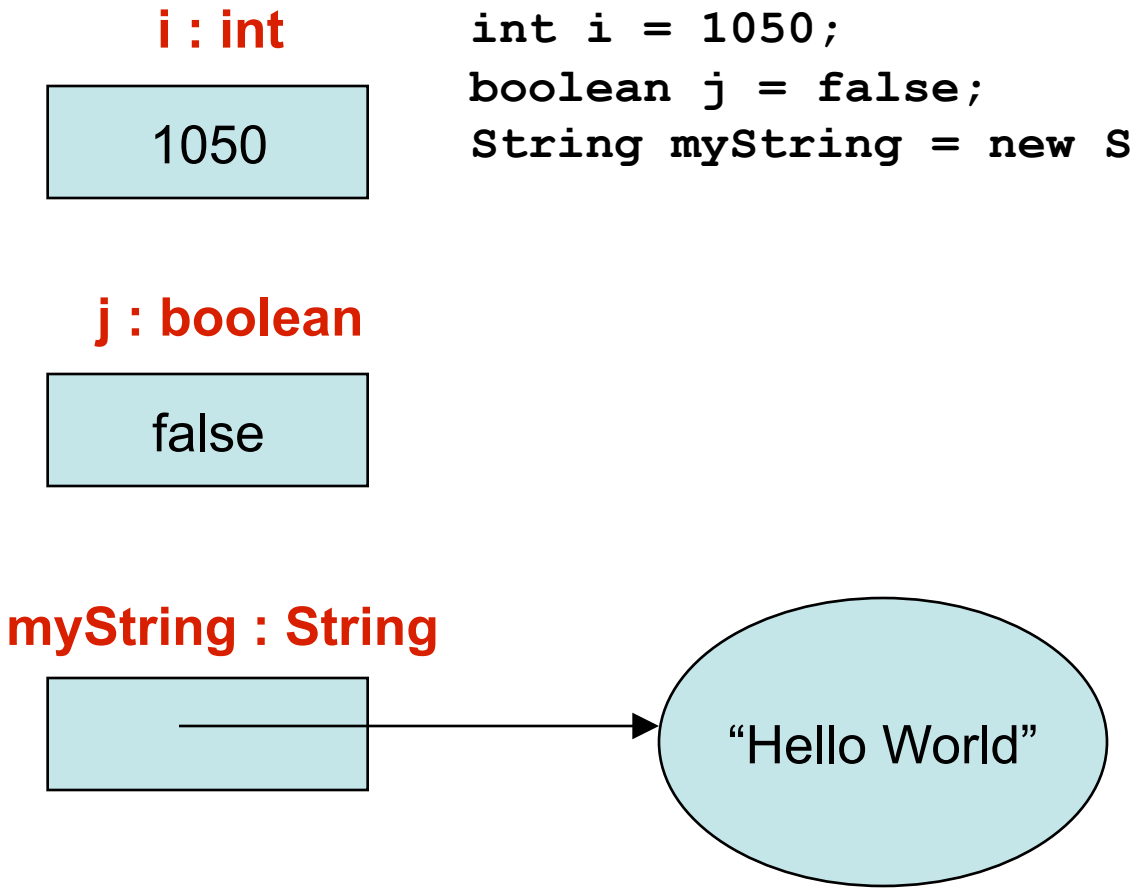
Objects

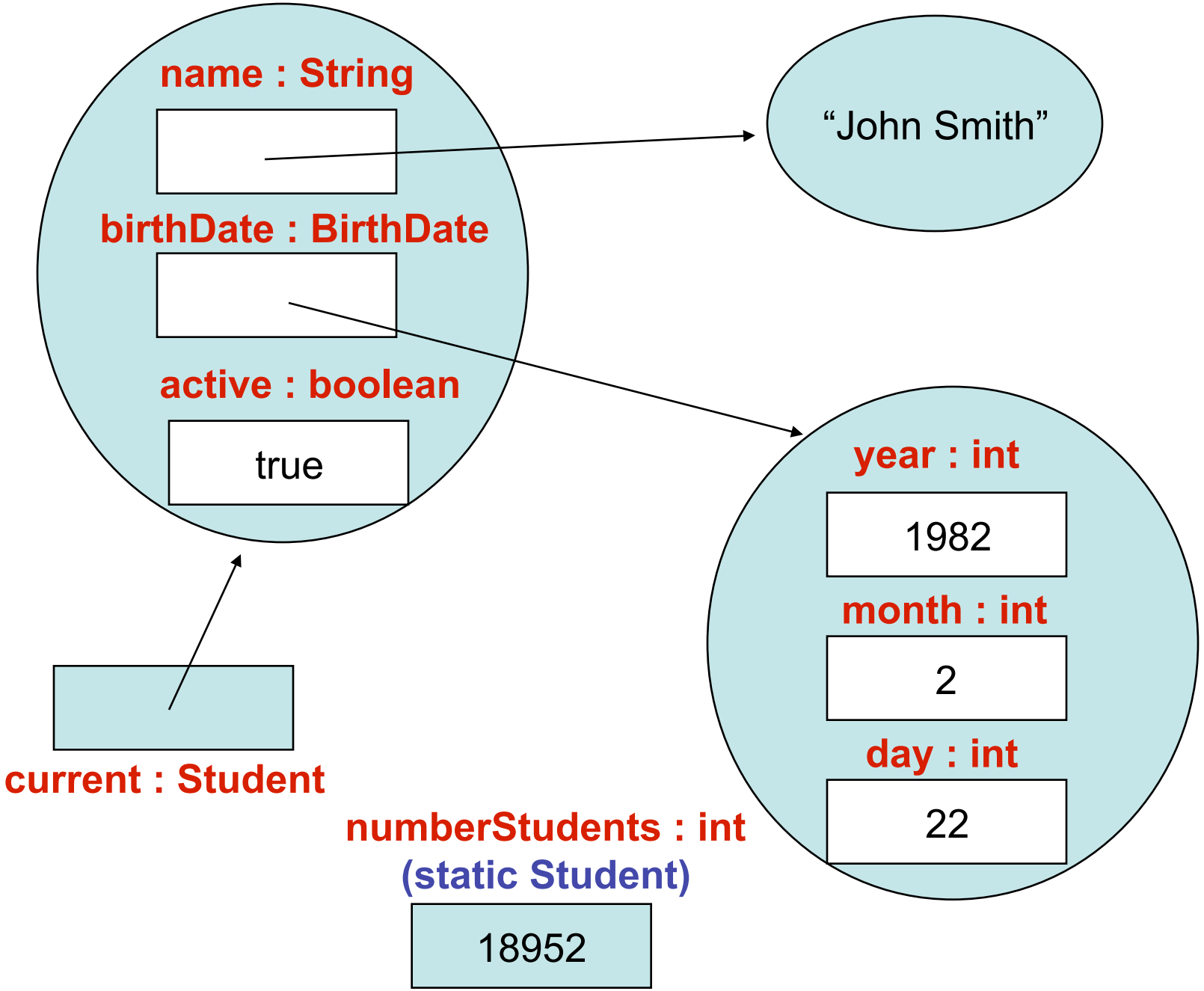
Drawn as circles, with internal boxes to represent data members

Strings are a 'special case' (see next slide)

Program:

```
int i = 1050;
boolean j = false;
String myString = new String("Hello World");
```




name : String
"John Smith"
birthDate : BirthDate
active : boolean
true
year : int
1982
month : int
2
day : int
22
current : Student
numberStudents : int
(static Student)
18952

Numeric Type Casting

Type Casting

Changes the type *(representation or encoding)* of a variable or constant

Narrowing Conversion

Convert from a larger range of values to a smaller range of values

- e.g. int x = (int) 5.32;

Widening Conversion

Convert from a smaller to a larger range of values

- e.g. double x = (double) 5;

Testing Reference Variable Values (==) vs. Object States (*.equals()*)

Equivalent References (==)

Tests whether the memory location referred to by reference variables is identical

(A == B): Does String variable A refer to the same memory location as String variable B?

Equivalent Object States (*.equals()*)

A method defined for the Object class, and overwritten by other Java classes (e.g. String) that normally tests for identical object states

(A.equals(B)): Does String variable A have the same state (characters) as String variable B?

WARNING: for Object class, equals() and == *are the same*

Variable Scope

Definition

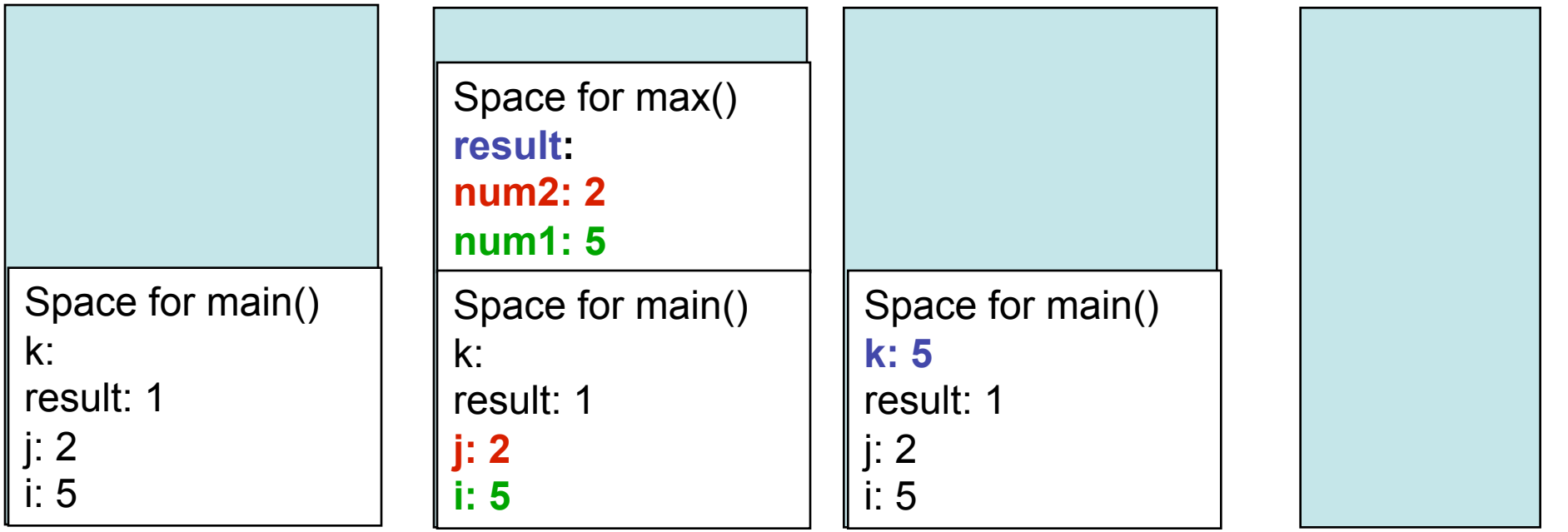
The program statements from which a variable may be referenced

Local variable

- A variable declared within a method
- May be referenced only in block in which they are declared
- Formal method parameters define local variables that may be referenced within the body of a method
  - Actual parameters (arguments) provide initial value for formal parameters (Java has a pass-by-value semantics for parameters)

Variable Masking or Shadowing

- Local variable definition "assumes the identity" (identifier) of variable in the parent scope
- References in the local scope are to the *locally declared* variable
- Local variables may mask instance or static variables in a method




```
public class TestMax {
  public static int max(int num1, int num2) {
        int result; …; return result;}

  public static void main(String[] args) {
    int i = 5, j = 2;
    int result = 1;
    int k = max(i, j);

    System.out.println("Max is " + k + " , result = " +
       result);
  }}
```

Organizing Variables: Arrays

Purpose

Allow us to organize variables in a structure

One-dimensional (1D) Arrays

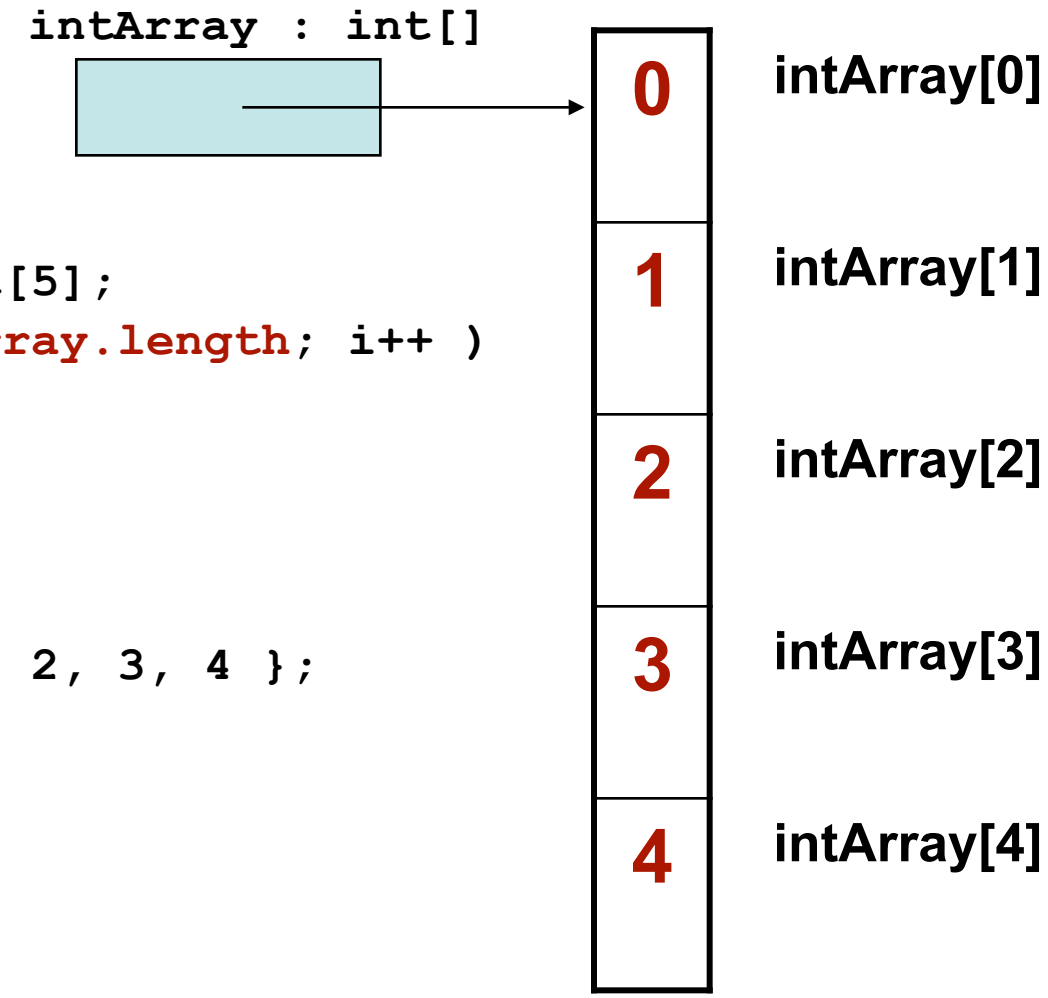
- A rectangular "box" containing "slots" for variables
  - Really a group of adjacent memory locations for storing variables
- Treated as a list with elements indexed starting from 0, up to array size – 1
    - Reason: array name represents location of first element; index is offset
    - *Eliminates the need to provide a different name for every variable in the array*
- Arrays may contain primitive or reference data types

1D Array: Primitive Data

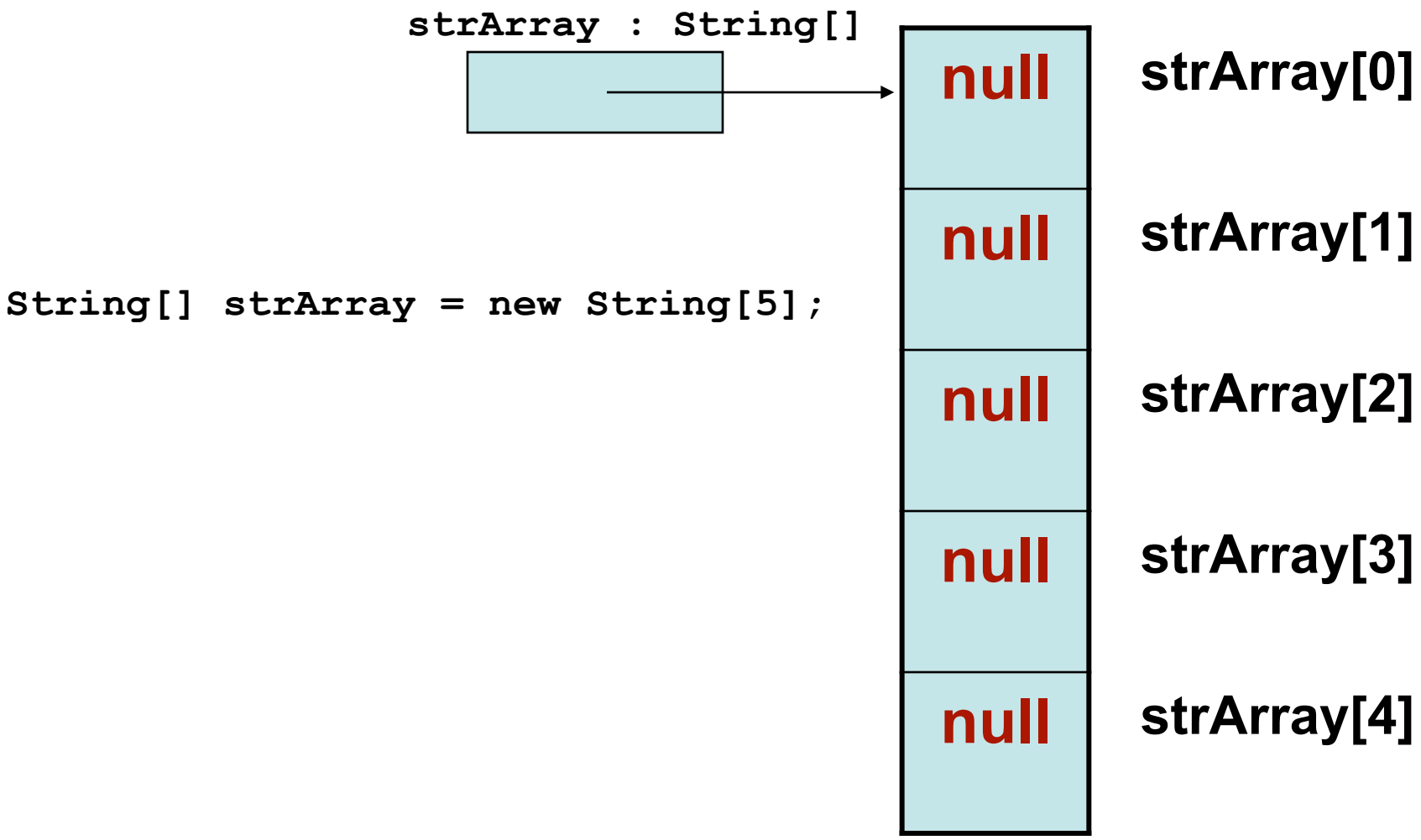
```
int intArray[] = new int[5];
for ( int i=0; i < intArray.length; i++ )
    intArray[i] = i;
```

vs.

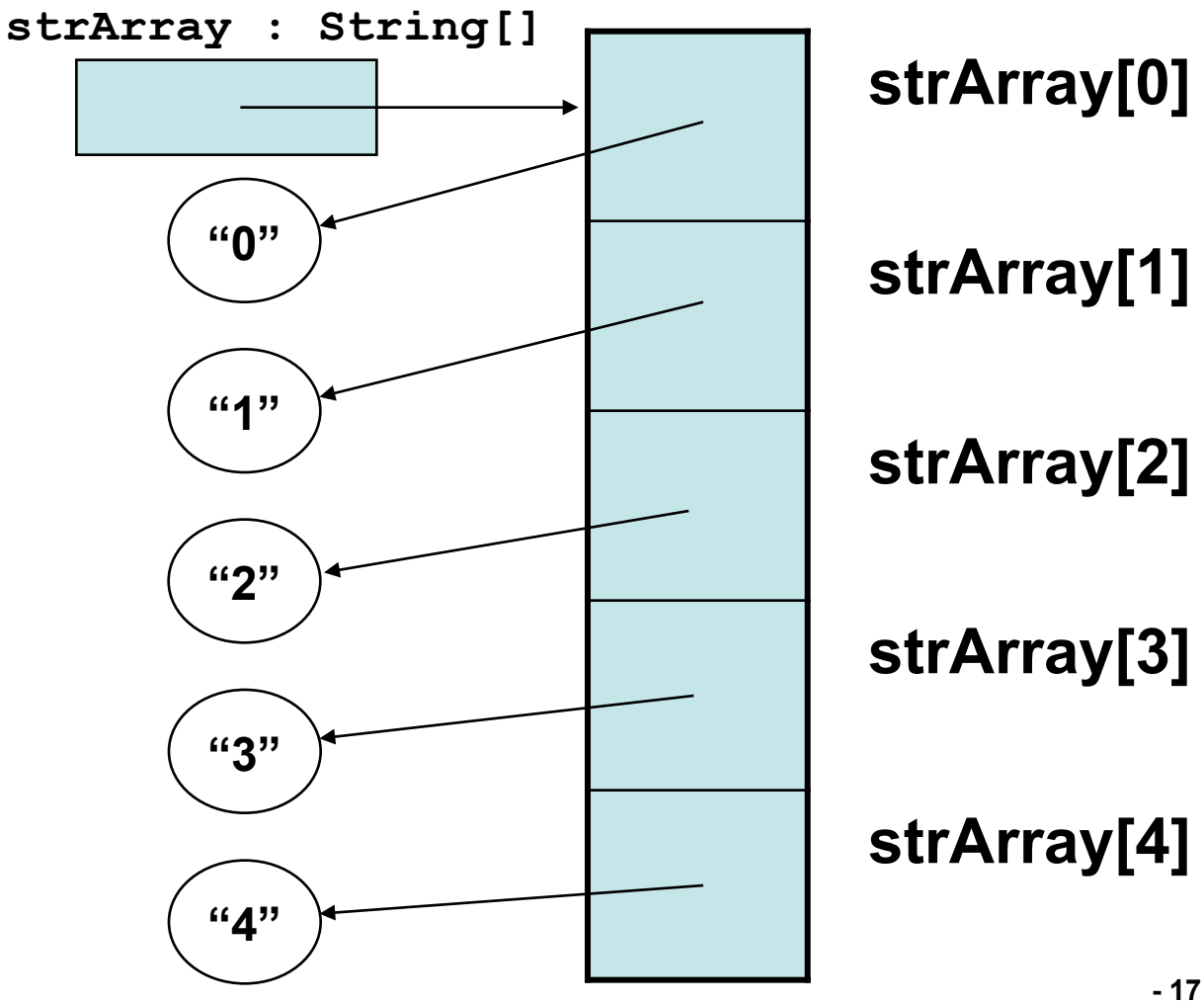
```
int intArray[] = { 0, 1, 2, 3, 4 };
```



1D Array: Reference Data



```
String[] strArray = new String[5];
for ( int i=0; i < strArray.length; i++)
    strArray[i] = new Integer(i).toString();
```



Mutli-Dimensional Arrays

Example

We might represent 150 marks as

– A one-dimensional array of 150 (double) floating point numbers

```
double[] marks = new double[150]; marks[0] = 95.1;
```

– A 2-D array of 15 (students) x 10 ((double) marks per student)

```
double[][] marks = new double[15][10]; marks[0][0]= 95.1;
```

– A 3-D array of 15 (students) x 5 (quizzes) x 2 ((double) mark for each section of each quiz, e.g. programming and short answer)

```
double[][][] marks = new double[15][5][2];
marks[0][0][0] = 95.1; marks[0][0][1]=85.0;
```

(Side Note) Ragged Arrays

Ragged Array Example

Array of 15 (students) x *different* sized arrays for each student, to represent the case where some students miss quizzes

- ```
  double[][] marks = new double[15][];
  marks[0] = new double[2];
  marks[1] = new double[3];
  ```

- Possible because java implements 2D and higher dimensional arrays as *arrays of arrays*
  ```

Control Flow

Definition

- The order in which statements in a program are executed
- "Simple" control flow: sequential execution of statements
  - "do a, then b, then c"

Conditional Statements (if, switch)

Change control flow by defining different *branches* of execution followed depending on Boolean conditions (expressions)

- "if C is true then do a, else do b"
- "if C is true then do {a, then b, then c}, else do {d, then e, then f}"

Iteration Statements (while, do...while, for)

Change control flow by repeating a statement or block (*compound statement*) while a Boolean condition holds

- "while C is true, do {a, then b, then c }"

Method Invocation

Produces a "jump" to the instructions in a method invocation

**also changes context (e.g. local variables): see earlier "TestMax" example on slide 14

What is '`this`' ?

Definition

- A reference to 'myself' for an object
- Used within instance methods for object invoking the method
- All instance variable references and method invocations implicitly refer to 'this'
  - Within an instance method: x = 2 *same as* this.x = 2; toString() *same as* this.toString())

Some Uses

1. Prevent masking of variables, e.g. formal params. and instance variables in a constructor:

```
public MyClass(int x){ this.x = x; }
```

2. Invoke other constructors within a class
   - Note: this(arg-list) must be first statement in constructor definition

```
public MyClass(int x){ this(); this.x = x; }
```

3. Have object pass itself as a method argument

```
someClass.printFancy(this);
```

Exercise: Variables

A. What are the four variable properties we discussed? Name and define each in point form.

B. Draw memory diagrams for the following.

1. int x = 5; double y = 2.0; y = x;
2. String s1 = “first”; String s2 = “second”; s1 = s2;
3. String strArray[ ] = { “a”, “b”, “c”, “d” };
4. boolean f[ ][ ] = new boolean[3][2];
5. int y[ ][ ] = { { 1, 2, 3 }, { 4, 5, 6, 7} };

C. Are the following legal? Why?

1. double y = (double) 5;
2. int x = 5.0;
3. int x = (int) 5.0;

Exercise: Variables (Cont'd)

What is output by the following program? How do the definitions for x and y differ? What kind of parameter and variable is args?

```
class SimpleExample {
    static int x = 5;
    int y = 2;

    public static void main(String args[ ]) {
        int x = 4;
        y = 9;
        System.out.println(x + y);
    }
}
```

Exercise: Methods

A. What is produced as output by the following?

```
int a = 2;
switch (a) {
    case 2: System.out.println("Case 2");
    case 1: System.out.println("Case 1"); break;
    default: System.out.println("Default Case"); }
```

C. Answer the following in 1-2 sentences each.

1. In what way are an *if* statement and a *while* statement the same?
2. How do an *if* statement and a *while* statement differ?
3. What extra elements are added to the conditional test in a *while loop* to produce a *for loop*?

Exercise: Methods, cont'd

C. What is wrong with the following?

```
class MethodExample {
    private int x = 5;
    static private int y = 3;

    public int methodOne() {return methodTwo();}
    public int methodOne(int x) {this.x = x; return x;}

    static public int methodTwo() {return y + methodOne(2); }
    static public int methodTwo(int x) { this.x = x; return x;}
}
```